Total Recursion over Lexicographical Orderings: Elementary Recursive Operators Beyond **PR**

David M. Cerna
Research Institute for Symbolic Computation (RISC)
cernadavid1@gmail.com

September 1, 2016

Abstract

In this work we generalize primitive recursion in order to construct a hierarchy of terminating total recursive operators which we refer to as *leveled primitive recursion of order* $i(\mathbf{PR}_i)$. Primitive recursion is equivalent to leveled primitive recursion of order 1 ($\mathbf{PR}_1$). The functions constructable from the basic functions make up $\mathbf{PR}_0$. Interestingly, we show that $\mathbf{PR}_2$ is a conservative extension of $\mathbf{PR}_1$. However, members of the hierarchy beyond $\mathbf{PR}_2$, that is $\mathbf{PR}_i$ where $i \geq 3$, can formalize the Ackermann function, and thus are more expressive than primitive recursion. It remains an open question which members of the hierarchy are more expressive than the previous members and which are conservative extensions. We conjecture that for all $i \geq 1$ $\mathbf{PR}_{2i} \subset \mathbf{PR}_{2i+1}$. Investigation of further extensions is left for future work.

1 Introduction

Primitive recursion dates back to [13, 18] when Dedekind introduced the concept of recursive. After Ackermann developed the function [1] carrying his name, it became obvious that the Dedekind's definition needed to be distinguished from the general concept of recursion. Dedekind's definition only handles recursion up to what we refer to now as primitive recursion. This distinction was solidified by Skolem [15]. Initially, Dedekind's primitive recursion represented what we know as terminating total recursive functions which are to be distinguished from functions requiring a μ-operator[12] to define. Though, such functions can still be total and terminating, they in general represent the class of partial functions, or more specifically computable functions.

To make a finer categorization of the concept of terminating total recursive functions without reliance on the class of partial functions, one needs to make a distinction concerning what objects the recursive operator is defined over. In defining system $\mathbf{T}$[11] Gödel typed the object over which the recursive operator operates. Thus, instead of producing a number the recursive operator can produce functions. One can think of this as producing $f^n(0)$ for some function f rather than $s^n(0)$ where $s(\cdot)$ is the successor function. Considering objects of type $Int \rightarrow Int$ is enough to express the Ackermann function, where Int is the type of the natural numbers.

One can go even further by typing objects using a stronger type theory. Girard introduced system **F**[10], which includes quantifiers over types. Though, the expressivity of system **F**, concerning functions over the natural numbers, is relatively the same as system **T**, system **F** can be used to construct inductive data types and has been shown equivalent to second order arithmetic.

The work presented here does not attempt to raise the bar of expressivity beyond system **F**, but rather attempts to find an elementary solution to the problem of expressing functions beyond primitive recursion, one that does not require iteration resulting in more complex objects than. Rather than using typed objects, we define recursive operators whose iteration is defined over a more complex ordering than the simple total ordering of the natural numbers. We chose the Lexicographical ordering of length n vectors of natural numbers being that it is the simplest extension of the total order of the natural numbers. In the simple type system used by system **T** this ordering would be represented by a right-associative string of Int.

One issue that still remains is to define how one gets from an arbitrary vector to the zero vector. We chose a rewrite system that treats the n^{th} position as an accumulator. We mean that every previous position can write a 1 (essentially subtracting 1 from itself to add to the last position) to the last position as a recursive step. Otherwise, any position i can write a 1 to the position $i + 1$ as a recursive step. For vectors of length 2 this results in an operator whose set of definable functions is equivalent to the definable functions of primitive recursion. However, the operator defined on vectors of length 3 has additional recursive steps not available to the primitive recursive operator. Essentially, using the accumulator we can treat the two other positions equally as if we are performing mutual recursion. This allows us to express the Ackermann function.

This novel system of recursive operators has direct application to schematically defined sequent calculus proofs [6, 9, 17], and the problem of cut elimination in the presence of induction. When an induction rule is present in the sequent calculus, cut elimination on the object level is not possible, but after a metalevel proof transformation the cuts can be eliminated, though, only when the end sequent is strong quantifier free. Though, this results in the loss of the subformula property of cut-free proofs. Using the CERES method [2, 3, 4] it is possible to perform cut-elimination in on recursively defined proofs without the loss of the subformula property[9]. Proof systems have been designed which deal with induction in an elegant way but at the loss of the subformula property [14]. It is difficult to perform cut-elimination on the recursively defined proof of [9]. This was shown in the work[7] where a simple mathematical statement required non-trivial analysis in order to perform cut-elimination, though the subformula property was retained. More complex statements than the one analysed in [7] seem to require a much more general recursive resolution calculus for a recursive definition of the refutation[6]. The goal of future work is to use the recursion we define here to develop this more general recursive resolution calculus. The recursive operators presented here seem to define a much more general class of inductionless induction[5, 8] than what has already been presented in literature, though we do not investigate this here and leave it for future work.

The rest of the paper is organized as follows. In Section 2 we discuss our generalization in detail. In Section 3 we provide a termination argument for $\mathbf{PR}_2$ and show that the functions definable in $\mathbf{PR}_2$ are equivalent to those definable in $\mathbf{PR}_1$. In Section 4 we show that $\mathbf{PR}_3$ is more expressive than

$\mathbf{PR}_2$. In Section 5 we show how to generalize to $\mathbf{PR}_i$ and prove that for $i \geq 3$ the Ackermann function is part of the definable functions. Finally, in Section 7 we discuss future work.

2 Generalizing Primitive Recursion

We generalize primitive recursion by generalizing the ordering the recursive operator is defined over, that is, we use a lexicographical ordering rather than the ordering of the natural numbers. This is significantly different than the construction of system $\mathbf{T}$ and $\mathbf{F}$, of which type the recursive operators. The ordering over which primitive recursion is defined is as follows:

$$x+1 > x > x-1 > \cdots > 0.$$

This results in the following rewrite rule defining the step case:

$$s(x) \Rightarrow x$$

and the normal form of 0. This is the ordering used to define $\mathbf{PR}_1$. For $\mathbf{PR}_2$ we define the recursion over the following ordering:

$$(x+1, y+1) > (x+1, y) > \cdots > (x+1, 0) > (x, z) > \cdots > (0, 0)$$

essentially, the lexicographical ordering of pairs. In comparison to $\mathbf{PR}_1$, it is not as clear, how to specify the rewrite rules in order to derive a proper recursion. We investigated the following two rules and show that they result in a recursive operator which is a conservative extension of $\mathbf{PR}_1$:

$$\begin{aligned}(x+1, y) &\Rightarrow (x, y+1)\\ (x, y+1) &\Rightarrow (x, y)\end{aligned}$$

Though, we do not gain any expressive power from our choice, the fact that $\mathbf{PR}_2$ is a conservative extension of $\mathbf{PR}_1$ justifies our choice of rewrite rules by showing that they indeed results in a recursive operator. However, unlike $\mathbf{PR}_1$, $\mathbf{PR}_2$ has branching as an implicit property of the operator. That is, from a step case of a $\mathbf{PR}_2$ function f, one can make a recursive call to both $f(x, y+2)$ and $f(x+1, y)$ from $f(x+1, y+1)$. Though, this does not result in greater expressive power, we will see, when defining $\mathbf{PR}_3$, how this leads to expressive power beyond $\mathbf{PR}_1$. We provide a graphical representation in Figure 1. The labels on the arrows show which member increases, decreases or stays the same. The nodes represent the members, where x is the first member and y is the second of the pair. The loop on y means the we can continuously follow the arrow till y is zero.

Now we discuss the extension of this concept to $\mathbf{PR}_3$. We extend the concepts used to define $\mathbf{PR}_2$ by add a third position, of which behaves similar to an accumulator. That is there are rewrite rules moving a 1 from the first or second position to the third. However, the first position can also write to the second. This results in a set of four rewrite rules defining the recursion over the

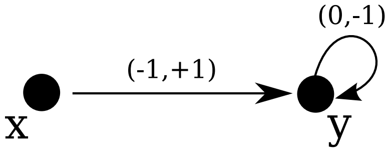

Figure 1: Graphical representation of the rewrite rules for $\mathbf{PR}_2$.

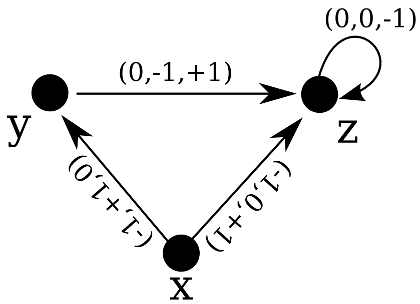

Figure 2: Graphical representation of the rewrite rules for $\mathbf{PR}_3$.

lexicographical ordering of triples:

$$\begin{aligned}(x+1,y,z) \Rightarrow & (x,y+1,z)\\ (x+1,y,z) \Rightarrow & (x,y,z+1)\\ (x,y+1,z) \Rightarrow & (x,y,z+1)\\ (x,y,z+1) \Rightarrow & (x,y,z)\end{aligned}$$

Notice that the first position is associated with two rewrite rules and the other two positions are associated with one rule. Further, the last two rules for $\mathbf{PR}_3$ are exactly the rules for $\mathbf{PR}_2$ and the last rule is exactly the rule from $\mathbf{PR}_1$. It is easy to see that from $\mathbf{PR}_i$ we get the rules for $\mathbf{PR}_{i+1}$ by adding two rules and thus, there are $2*(i-1)$ rewrite rules for $\mathbf{PR}_i$. We summarize the rewrite system graphically in Figure 2. The meaning of the components of the graphical represent is the same as in Figure 1. Figure 3 illustrates the graphical representation for the case of i variables.

What we have not discussed yet is how functions are constructed from these orderings. Similar to primitive recursion, a $\mathbf{PR}_i$ function f will have several cases, and each of them has a $\mathbf{PR}_i$ function which takes the recursive calls to f as arguments. The difference is that f will have many more cases than primitive recursion, 2^i to be precise. For $\mathbf{PR}_1$ we have two cases, which matches primitive recursion, the step case and the base case. We distinguish between the recursive variable being positive or zero. For $\mathbf{PR}_i$ we make the same distinction except we have to consider all combinations of the i variables. When no variable is set to zero, we refer to that case as the *step case*, when at least one variable is set to zero, we will consider that case to be a *soft-basecase* and when all variables are set to zero we refer to the case as a *hard-basecase*. The number of recursive calls allowed at each soft-basecase is dependent on the number of variables which

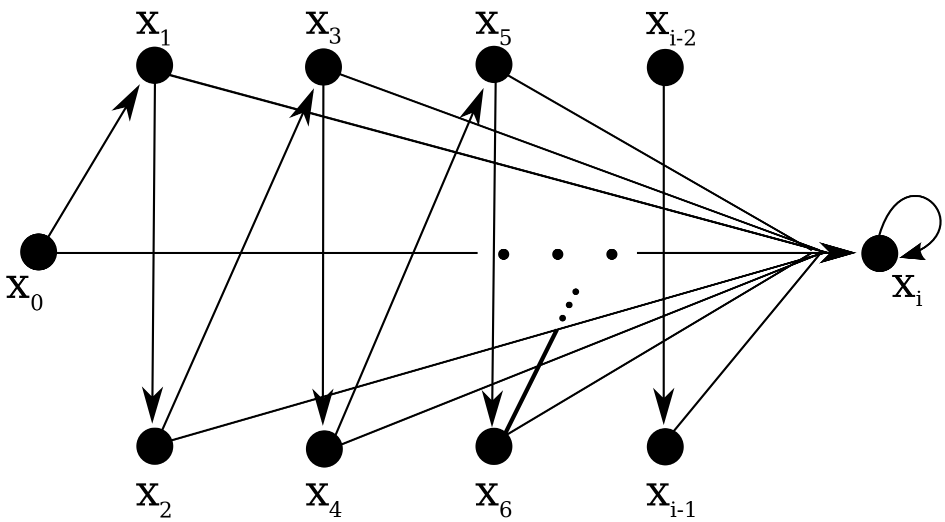

Figure 3: Graphical representation of the rewrite rules for $\mathbf{PR}_i$.

have a positive value. If the value of a variable is zero we cannot substract 1 from it, a requirement of every rewrite rule. Thus, when all variables are positive we can make $2(i-1)$ recursive calls. This will make defining $\mathbf{PR}_i$ a bit cumbersome, but it can be elegantly organized.

3 Termination of $\mathbf{PR}_2$ and equivalence to $\mathbf{PR}_1$

We begin by showing that $\mathbf{PR}_2$ always terminates at $(0,0)$ and that any function of $\mathbf{PR}_2$ can be constructed in $\mathbf{PR}_1$ and any $\mathbf{PR}_1$ function can be constructed in $\mathbf{PR}_2$.

Definition 1. *We define the function $f \in \mathbf{PR}_0$ if f is constructed from the constant function 0, the successor function $s(\cdot)$, function composition, and projection functions.*

Lemma 1. $\mathbf{PR}_0 \subset \mathbf{PR}_1$

Proof. By definition. $\square$

Definition 2. *Let $x, k \geq 0$, $g, h, n_1, \cdots, n_k \in \mathbf{PR}_1$. Then we define the function $f \in \mathbf{PR}_1[g,h](x)$ as follows:*

$$\begin{aligned} f(n_1, \cdots, n_k, x+1) &\Rightarrow g(n_1, \cdots, n_k, x, f(n_1, \cdots, n_k, x)) \\ f(n_1, \cdots, n_k, 0) &\Rightarrow h(n_1, \cdots, n_k) \end{aligned}$$

Theorem 1. *Every $f \in \mathbf{PR}_1$ terminates.*

Proof. $\mathbf{PR}_1$ is equivalent to PR which is known to terminate. $\square$

Definition 3. *Let* $x, y, k \geq 0$, $g^3, g^2, g^1, g^0, \overline{n}_k = n_1, \cdots, n_k \in \mathbf{PR}_2$. *Then we define the function* $f \in \mathbf{PR}_2\left[g^3, g^2, g^1, g^0\right](x, y)$ *as follows:*

$$\begin{aligned}
f(\overline{n}_k, x+1, y+1) \Rightarrow & g^3(\overline{n}_k, x, y, f(\overline{n}_k, x, y+2), f(\overline{n}_k, x+1, y)) \\
f(\overline{n}_k, x+1, 0) \Rightarrow & g^2(\overline{n}_k, x, f(\overline{n}_k, x, 1)) \\
f(\overline{n}_k, 0, y+1) \Rightarrow & g^1(\overline{n}_k, y, f(\overline{n}_k, 0, y)) \\
f(\overline{n}_k, 0, 0) \Rightarrow & g^0(\overline{n}_k)
\end{aligned}$$

Theorem 2. $\mathbf{PR}_2$ *is terminating at* $(0, 0)$.

Proof. We know our rewrite rules are defined over the lexicographical ordering of pairs. Also, any call to one of our rewrite rules results in a pair lower in the ordering than the starting pair. As long as a pair has a positive value we can apply a rewrite rule and thus we always reach $(0, 0)$. □

Theorem 3. *For every function* $f \in \mathbf{PR}_1$ *there exists a function* $f' \in \mathbf{PR}_2$ *such that for all* $x \geq 0$ *there exists* $y, z \geq 0$*, where* $x = y + z$*, such that* $f(\overline{n}_k, x) = f'(\overline{n}_k, y, z)$.

Proof. Let f be the function:

$$\begin{aligned}
f(\overline{n}_k, x+1) \Rightarrow & g(\overline{n}_k, x, f(\overline{n}_k, x)) \\
f(\overline{n}_k, 0) \Rightarrow & h(\overline{n}_k)
\end{aligned}$$

Then we define the function f' as follows:

$$\begin{aligned}
f(\overline{n}_k, x+1, y+1) \Rightarrow & f(\overline{n}_k, x, y+2) \\
f(\overline{n}_k, x+1, 0) \Rightarrow & f(\overline{n}_k, x, 1) \\
f(\overline{n}_k, 0, y+1) \Rightarrow & g(\overline{n}_k, y, f(\overline{n}_k, 0, y)) \\
f(\overline{n}_k, 0, 0) \Rightarrow & h(\overline{n}_k)
\end{aligned}$$

□

Theorem 4. *For every function* $f \in \mathbf{PR}_2$ *there exists functions* $A, B \in \mathbf{PR}_1$ *such that for all* $x, y \geq 0$, $f(\overline{n}_k, x, y) = A(\overline{n}_k, x, y)$.

Proof. Let f be the function:

$$\begin{aligned}
f(\overline{n}_k, x+1, y+1) \Rightarrow & g^3(\overline{n}_k, x, y, f(\overline{n}_k, x, y+2), f(\overline{n}_k, x+1, y)) \\
f(\overline{n}_k, x+1, 0) \Rightarrow & g^2(\overline{n}_k, x, f(\overline{n}_k, x, 1)) \\
f(\overline{n}_k, 0, y+1) \Rightarrow & g^1(\overline{n}_k, y, f(\overline{n}_k, 0, y)) \\
f(\overline{n}_k, 0, 0) \Rightarrow & g^0(\overline{n}_k)
\end{aligned}$$

Let A be the function:

$$
\begin{aligned}
A(\overline{n}_k, x+1, y) \Rightarrow & \textbf{If}(y > 0) \textbf{ Then} \\
& \quad g^3(\overline{n}_k, x, y, A(\overline{n}_k, x, y+1), B(\overline{n}_k, y-1, x+1)) \\
& \textbf{Else} \\
& \quad g^2(\overline{n}_k, x, A(\overline{n}_k, x, 1)) \\
A(\overline{n}_k, 0, y) \Rightarrow & \textbf{If}(y > 0) \textbf{ Then} \\
& \quad g^1(\overline{n}_k, y, B(\overline{n}_k, y-1, 0)) \\
& \textbf{Else} \\
& \quad g^0(\overline{n}_k)
\end{aligned}
$$

Let B be the function:

$$
\begin{aligned}
B(\overline{n}_k, y+1, x) \Rightarrow & \textbf{If}(x > 0) \textbf{ Then} \\
& \quad g^3(\overline{n}_k, x, y, A(\overline{n}_k, x-1, y+2), B(\overline{n}_k, y, x)) \\
& \textbf{Else} \\
& \quad g^1(\overline{n}_k, x, B(\overline{n}_k, y, 0)) \\
B(\overline{n}_k, 0, x) \Rightarrow & \textbf{If}(x > 0) \textbf{ Then} \\
& \quad g^1(\overline{n}_k, x, A(\overline{n}_k, x-1, 1)) \\
& \textbf{Else} \\
& \quad g^0(\overline{n}_k)
\end{aligned}
$$

This finalizes the proof. □

4 $\mathbf{PR}_3$ is More Expressive Than $\mathbf{PR}_1$

Now we show that $\mathbf{PR}_3$ is more expressive than $\mathbf{PR}_1$ by formalizing the the Ackermann function in $\mathbf{PR}_3$. This shows that $\mathbf{PR}_3$ is more expressive than $\mathbf{PR}_1$ because $\mathbf{PR}_1$ is equivalent to primitive recursion and primitive recursion cannot express the Ackermann function. Before we get to the formalization we define $\mathbf{PR}_3$.

Definition 4. *Let* $x, y, z, k \geq 0$, $g^7, g^6, g^5, g^3, g^2, g^1, g^0, \overline{n}_k = n_1, \cdots, n_k \in \mathbf{PR}_3$. *Then we define the function* $f \in \mathbf{PR}_3\left[g^7, g^6, g^5, g^3, g^2, g^1, g^0\right](x, y, z)$ *as fol-*

lows:

$$
\begin{aligned}
f(\overline{n}_k, x+1, y+1, z+1) \Rightarrow & g^7(\overline{n}_k, x, y, z, f(\overline{n}_k, x, y+1, z+2), f(\overline{n}_k, x, y+2, z+1), \\
& f(\overline{n}_k, x+1, y, z+2), f(\overline{n}_k, x+1, y+1, z)) \\
f(\overline{n}_k, x+1, y+1, 0) \Rightarrow & g^6(\overline{n}_k, x, y, f(\overline{n}_k, x, y+1, 1), f(\overline{n}_k, x, y+2, 0), \\
& f(\overline{n}_k, x+1, y, 1)) \\
f(\overline{n}_k, x+1, 0, z+1) \Rightarrow & g^5(\overline{n}_k, x, z, f(\overline{n}_k, x, 0, z+2), f(\overline{n}_k, x, 1, z+1), \\
& f(\overline{n}_k, x+1, 0, z)) \\
f(\overline{n}_k, x+1, 0, 0) \Rightarrow & g^4(\overline{n}_k, x, f(\overline{n}_k, x, 0, 1), f(\overline{n}_k, x, 1, 0)) \\
f(\overline{n}_k, 0, y+1, z+1) \Rightarrow & g^3(\overline{n}_k, y, z, f(\overline{n}_k, 0, y, z+2), f(\overline{n}_k, 0, y+1, z)) \\
f(\overline{n}_k, 0, y+1, 0) \Rightarrow & g^2(\overline{n}_k, y, f(\overline{n}_k, 0, y, 1)) \\
f(\overline{n}_k, 0, 0, z+1) \Rightarrow & g^1(\overline{n}_k, z, f(\overline{n}_k, 0, 0, z)) \\
f(\overline{n}_k, 0, 0, 0) \Rightarrow & g^0(\overline{n}_k)
\end{aligned}
$$

Theorem 5. $\mathbf{PR}_3$ *is terminating at* $(0, 0, 0)$.

Proof. The rewrite rules follow the lexicographical ordering. □

Theorem 6. $\mathbf{PR}_3$ *is more expressive than* $\mathbf{PR}_2$.

Proof. The Ackermann function can be expressed in $\mathbf{PR}_3$ and cannot be expressed in $\mathbf{PR}_2$ because $\mathbf{PR}_2$ is equivalent to $\mathbf{PR}_1$. The Ackermann function is expressed by the following function A:

$$
\begin{aligned}
A(x+1, y+1, z+1) \Rightarrow & A(x, A(x+1, y, z+2), z) \\
A(x+1, y+1, 0) \Rightarrow & A(x, A(x+1, y, 1), 0) \\
A(x+1, 0, z+1) \Rightarrow & A(x, 1, z) \\
A(x+1, 0, 0) \Rightarrow & A(x, 1, 0) \\
A(0, y+1, z+1) \Rightarrow & y+2 \\
A(0, y+1, 0) \Rightarrow & y+2 \\
A(0, 0, z+1) \Rightarrow & 1 \\
A(0, 0, 0) \Rightarrow & 1
\end{aligned}
$$

This can be written in simpler terms by combining similar cases. After combining the similar cases the function resembles the Ackermann function.

$$
\begin{aligned}
A(x+1, y+1, z) \Rightarrow & A(x, A(x+1, y, z+1), z) \\
A(x+1, 0, z) \Rightarrow & A(x, 1, z) \\
A(0, y, z) \Rightarrow & y+1
\end{aligned}
$$

□

Notice that the Ackermann function is expressible in a very weak fragment of $\mathbf{PR}_3$, that is, none of the branching mechanisms available in $\mathbf{PR}_3$ are used.

However, we do use the “branching” in the sense that two of the possible branching recursive calls show up in the formal definition, though, in separate cases. Also, the formalization takes advantage of the accumulation properties of the third variable. This allows us to change the value of x and y in a mutually recursive way while maintain termination.

5 Beyond $\mathbf{PR}_3$

Theorem 7. *For all $i > 3$, $\mathbf{PR}_i$ is terminating at $(\overbrace{0, \cdots, 0}^{i\ times})$.*

Proof. The rewrite rules follow the lexicographical ordering. □

5.1 Formalization of Leveled Primitive Recursion Beyond Order 3

It is possible to generalize leveled primitive recursion beyond order 3 by adding more recursive variables. However, the hierarchy gets increasingly more complex to present. One way of presenting order beyond 3 is to use functions instead of vectors, though the codomain of the functions still follow the lexicographical ordering allowing use to order the functions. Let us consider the set $\mathbb{F}_n$ of all total functions $f : \{1, \cdots, n\} \to \mathbb{N} \cup \{0\}$ for $n \in \mathbb{N}$. We can define our recursive operators as a set unary operators $\textcircled{n}^j_k : \mathbb{F}_n \to \mathbb{F}_n$ for $1 \leq j < n$ and $k \in \{j + 1, n\}$, where j is the position which we are subtracting from and k is the position we are adding too. The operator $\textcircled{n}^n : \mathbb{F}_n \to \mathbb{F}_n$ is special in that it only subtracts from the n^{th} position. We will refer to the function $f^0_n \in \mathbb{F}_n$ as the zero function, that is $\forall i \in \{1, \cdots, n\}\ f^0_n(i) = 0$.

$$
f\textcircled{n}^j_k = \begin{cases}
f^0_n & f = f^0_n \\
\left\{ f' \;\middle|\; \begin{array}{c} f'(j) = f(j) - 1, \\ f'(k) = f(k) + 1, \\ \textit{and } \forall i \in \{1, \cdots, n\} \\ (i \neq j \land i \neq k \to f'(i) = f(i)) \end{array} \right\} & f(j) \neq 0 \\
f\textcircled{n}^r_n & \begin{array}{c} r = \min\limits_{i \in \{1, \cdots, n\}} \{f(i) > 0\} \\ r < n \end{array} \\
f\textcircled{n}^n & \textit{otherwise}
\end{cases}
$$

$$
f\textcircled{n}^n = \begin{cases}
f^0_n & f = f^0_n \\
\left\{ f' \;\middle|\; \begin{array}{c} f'(n) = f(n) - 1, \\ \textit{and } \forall i \\ (i < n \to f'(i) = f(i)) \end{array} \right\} & f(n) \neq 0 \\
f\textcircled{n}^r_n & \begin{array}{c} r = \min\limits_{i \in \{1, \cdots, n\}} \{f(i) > 0\} \\ r < n \end{array}
\end{cases}
$$

$$f \downarrow_n = \{f' | \forall i((f(i) = 0 \rightarrow f'(i) = f(i)) \wedge (f(i) > 0 \rightarrow f'(i) = f(i) - 1)))\}$$

The last function we need to define before we simplify the definition of the **PR** Hierarchy is a function mapping $\mathbb{F}_n$ to natural numbers. The function is defined as follows:

$$\Sigma_n(f, i) = 2^{\sigma(f(i)) \cdot (n-i)} + \Sigma_n(f, i-1)$$
$$\Sigma_n(f, 1) = 2^{\sigma(f(i)) \cdot (n-i)}$$

where

$$\sigma(i) = \begin{cases} 1 & i > 0 \\ 0 & i = 0 \end{cases}$$

Now we give the succinct definition of the **PR** Hierarchy:

Definition 5. *Let $n \geq 0$, $f \in \mathbb{F}_n$, $g^{2^i}, g^{2^i-1}, \cdots, g^1, g^0, \overline{n}_k = n_1, \cdots, n_k \in \mathbf{PR}_i$. Then we define the function $h \in \mathbf{PR}_i \left[g^{2^i}, g^{2^i-1}, \cdots, g^1, g^0\right](f)$ as follows:*

$$\begin{aligned} h(\overline{n}_k, f) \Rightarrow & g^{\Sigma_n(f,n)}(\overline{n}_k, f \downarrow_n (0), \cdots, f \downarrow_n (n), h(\overline{n}_k, f \textcircled{n}^1_2), h(\overline{n}_k, f \textcircled{n}^1_n), \\ & h(\overline{n}_k, f \textcircled{n}^2_3), \cdots, h(\overline{n}_k, f \textcircled{n}^{n-1}_n), h(\overline{n}_k, f \textcircled{n}^n)) \\ h(\overline{n}_k, f^0_n) \Rightarrow & g^0(\overline{n}_k) \end{aligned}$$

Theorem 8. *For all $i > 3$, $\mathbf{PR}_i$ is terminating at f^0_i.*

Proof. The function transformations follow the lexicographical ordering. □

Theorem 9. *For all $n \geq 3$, $\mathbf{PR}_n$ is more expressive than $\mathbf{PR}_2$.*

Proof. We can prove this theorem by showing that the Ackermann function can be formalized in every $\mathbf{PR}_i$. This can easily be done by setting all the variables greater than the last three to zero. An additional function is needed to define the Ackermann function using the above definition.

$$f'_f(x) \Rightarrow \begin{cases} 0 & 1 \leq x < n-2 \\ f \downarrow_n (n-2) & x = n-2 \\ A(f \textcircled{n}^n_{n-1}) & x = n-1 \\ f \downarrow_n (n) & x = n \end{cases}$$

The Ackermann function is as follows:

$$A(f) \Rightarrow \begin{cases} A(f \textcircled{n}^1_n) & 7 < \Sigma_n(f, n) \\ A(f'_f(x)) & 6 \leq \Sigma_n(f, n) \leq 7 \\ A(f \textcircled{n}^{n-1}_{n-2}) & 4 \leq \Sigma_n(f, n) \leq 5 \\ y + 2 & 2 \leq \Sigma_n(f, n) \leq 3 \\ 1 & 0 \leq \Sigma_n(f, n) \leq 1 \end{cases}$$

□

6 The Mutual Ackermann Function

In $\mathbf{PR}_5$, a novel and interesting emergent property shows up, what we refer to as the mutual Ackermann function. We provide the following formal definition of the function, though we do not provide all 32 cases, but rather provide the succinct version.

$$\begin{aligned}
A(x+1, y+1, z+1, w+1, r) \Rightarrow & A(x, A(x+1, y, z+1, w+1, r+1), z, \\
& A(x+1, y+1, z+1, w, r+1), r) \\
A(x+1, 0, z+1, w+1, r) \Rightarrow & A(0, A(x, 1, z+1, w+1, r), z, \\
& A(x+1, 0, z+1, w, r+1), r) \\
A(0, y, z+1, w+1, r) \Rightarrow & A(0, y, z, A(0, y, z+1, w, r+1), r) \\
A(x+1, y+1, z+1, 0, r) \Rightarrow & A(x, A(x+1, y, z+1, w+1, r+1), z, \\
& A(x+1, y+1, z+1, 1, r), r) \\
A(x+1, y+1, 0, w, r) \Rightarrow & A(x, A(x+1, y, 0, w, r+1), 0, w, r) \\
A(x+1, 0, z+1, 0, r) \Rightarrow & A(x, 1, z+1, w+1, r) + A(x+1, 0, z, 1, r) \\
A(x+1, 0, 0, w, r) \Rightarrow & A(x, 1, 0, w, r) + w \\
A(0, y, z+1, 0, r) \Rightarrow & A(0, y, z, 1, r) + y \\
A(0, y, 0, w, r) \Rightarrow & y + w + 1
\end{aligned}$$

A similar function can be constructed for all $\mathbf{PR}_{2n+1}$, where $n > 1$. A note on the growth rate, $A(2,1,0,0,0) = A(2,1,0) = 5$ and $A(0,0,1,1,0) = A(1,1,0) = 3$, but $A(2,1,1,1,0) = 25556$. This illustrates that the mutual Ackermann function is just two Ackermann functions call mutually. Note that this is only the first function in the mutual Ackermann function Hierarchy. We conjecture that this function grows faster than any $\mathbf{PR}_3$ function in a similar way to the Ackermann function growing faster than any $\mathbf{PR}_1$ function. We also conjecture a relationship between these functions and ε numbers. This would imply it is beyond **PA**!!

7 Conclusion & Furture Work

There are many properties of these recursive operators which has not been examined in this introductory work. For example, The relationship between this work and System **T** and **F** has not be investigated. Also, our using one variable as an accumulator can easily be extended to a set of accumulator variables. Essentially, this would result in a secondary recursion constructed by the first recursion. This would imply the possibility of generating recursive functions based on the behaviour of a previous recursive function. Also, these properties can lead to alternative formalizations of functions. Such constructions are related to the type theory founding systems **T** and **F**. However, unlike those systems, terminating extensions of our construction are easy to develop. Termination is a trivial property of our recursive operators and many of its generalizations. Another direction yet to be discussed is recursion terminating at a countably infinite time interval. This might seem to resemble the halting problem, but

as it has been shown, even infinite time Turing machines have their own halting problem [16] and termination is not well defined for the halting problem in the first case. Though, such functions are of little practical value, we can discuss recursive constructibility of certain infinite sets in an alternative ways to induction. We will address many of these issues in future work.

One very important question which remains open is whether $\mathbf{PR}_i$ is strictly contained in $\mathbf{PR}_{i+1}$. Like the case of $\mathbf{PR}_1$ and $\mathbf{PR}_2$ it my be the case that other members of the hierarchy are equivalent. We conjecture that what $\mathbf{PR}_{2n}$ is strictly contained in $\mathbf{PR}_{2n+1}$. There is some justification for this conjecture, $2n+1$ variables is enough to run n mutual Ackermann functions where the last variable acts as the accumulator. One cannot compute such a function with $2n$. Though this would imply the strict inclusion of $\mathbf{PR}_{2n}$ in $\mathbf{PR}_{2n+1}$ it does not provide any answer to the question of strict inclusion of $\mathbf{PR}_{2n+1}$ in $\mathbf{PR}_{2n}$. This case remains open.

References

[1] Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Math. Ann.*, 99:118–133, 1928.

[2] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Proof transformation by ceres. In Jonathan M. Borwein and William M. Farmer, editors, *Mathematical Knowledge Management (MKM) 2006*, volume 4108 of *Lecture Notes in Artificial Intelligence*, pages 82–93. Springer, 2006.

[3] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Ceres: An analysis of Fürstenberg's proof of the infinity of primes. *Theoretical Computer Science*, 403(2-3):160–175, August 2008.

[4] Matthias Baaz and Alexander Leitsch. Cut-elimination and redundancy-elimination by resolution. *Journal of Symbolic Computation*, 29:149–176, 2000.

[5] A. Bundy. The automation of proof by mathematical induction. *Handbook of Automated Reasoning*, pages 845–911, 2001.

[6] David M. Cerna. *Advances in schematic cut elimination.* PhD thesis, Technical University of Vienna, 2015. http://media.obvsg.at/p-AC12246421-2001.

[7] David M. Cerna and Alexander Leitsch. *Schematic Cut Elimination and the Ordered Pigeonhole Principle*, pages 241–256. Springer International Publishing, Cham, 2016.

[8] Hubert Comon. Inductionless induction. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 913–962. Elsevier and MIT Press, 2001.

[9] Cvetan Dunchev, Alexander Leitsch, Mikheil Rukhaia, and Daniel Weller. Cut-elimination and proof schemata. In *Revised Selected Papers of the 10th International Tbilisi Symposium on Logic, Language, and Computation -*

Volume 8984, pages 117–136, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

[10] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

[11] Kurt Gödel. Über eine bisher noch nicht benützte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958.

[12] Stephen Kleene. Introduction to Metamathematics, 1952.

[13] Franz Lemmermeyer. Richard dedekind: "was sind und was sollen die zahlen?". *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 115(2):115–117, 2013.

[14] Raymond Mcdowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:2000, 1997.

[15] Thoralf Skolem. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, chapter The foundations of elementary arithmetic established by means of the recursive mode of thought, without the use of apparent variables ranging over infinite domains, pages 302–333. Harvard University Press, 2002.

[16] Apostolos Syropoulos. *Hypercomputation: Computing Beyond the Church-Turing Barrier*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[17] Gaisi Takeuti. *Proof Theory*, volume 81 of *Studies in logic and the foundations of mathematics*. American Elsevier Pub., 1975.

[18] George Tourlakis. *Lectures in Logic and Set Theory: Volume 1, Mathematical Logic*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2010.